2. Diagramme de classes

2.1 Définition diagramme de classes

C'est un diagramme qui montre une collection d'éléments statiques (classes), leur contenu et les relations entre eux.

Un diagramme de classes fait abstraction des aspects dynamiques et temporels.

Dans le cas d'un modèle complexe, plusieurs diagrammes de classes complémentaires peuvent être construits, en association avec des cas d'utilisation, un scénario précis ou un paquetage.

Pour préciser un cas complexe, un diagramme de classes peut être instancié en diagrammes d'objets

Le diagramme de classes provient principalement de OMT RUMBAUGH et de OOD BOOCH

En analyse, le diagramme de classe représente la structure des informations manipulées par les utilisateurs. En conception, il représente la structure d'un code orienté objet.

2.2 Définition d'une classe

Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des objets ayant ces propriétés.

Classe = attributs + méthodes + instanciation

Les classes se désignent par un **nom**, contiennent des **attributs** et des **méthodes** associées

♦ les **attributs** sont des propriétés caractéristiques de la classe. Les attributs peuvent être privés, publics, protégés. Ils sont le plus souvent privés. Les classes respectent le principe d'encapsulation des données.

♦ les **méthodes** sont des procédures spécifiques à une classe. Elles sont le plus souvent publiques. Elles peuvent être privées : on parle dans ce cas de méthodes d'implémentation

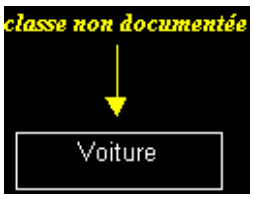
Plusieurs niveaux de représentation sont possibles pour une classe, selon la visibilité recherchée.

♦ la **visibilité** permet de définir la manière dont chaque attribut et chaque méthode de classe peut être vue par les autres classes. Elle se définie en quatre niveaux :

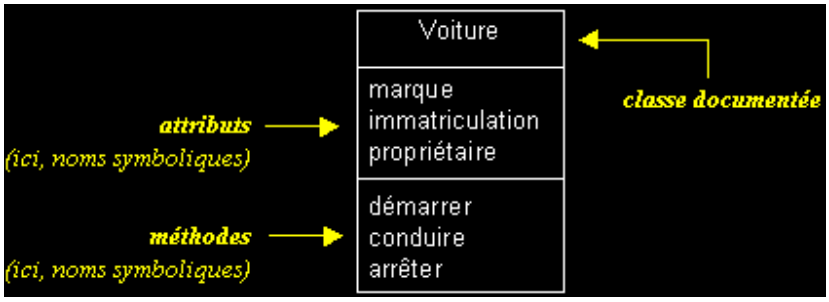
+ public	visible par toutes les classes
- privé	visible seulement par la classe
# protégé	visible par la classes et les classes héritées
* package	visible par les classes du package

Exemple :

Niveau 1 : classe non documentée



Niveau 2 : classe documentée avec attributs et méthodes



Niveau 3 : classe détaillée présentant les niveaux de protection

Les attributs sont typés, les prototypes des méthodes sont spécifiés et les niveaux de protection (+ : public, - : privé) des composants sont spécifiés

voiture	
+ marque	: char
+ immatriculation	: char
+ couleur	: char
- puissanceFiscale	: int
- poidsVide	: int
+ dateFabrication	: java.util.Date
+ propriétaire	: char
+ Demarrer ()	
+ arreter ()	
+ conduire (char de, char a)	
- vendre (int prix)	

2.3 Associations entre classes

2.3.1 Définition

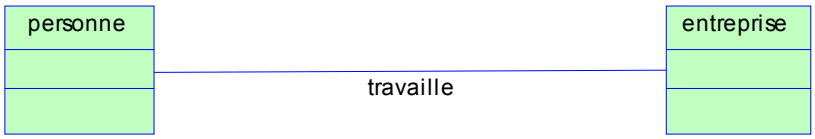
Une association exprime une connexion sémantique bidirectionnelle entre deux classes.

L'association binaire est représentée par un trait entre les deux classes

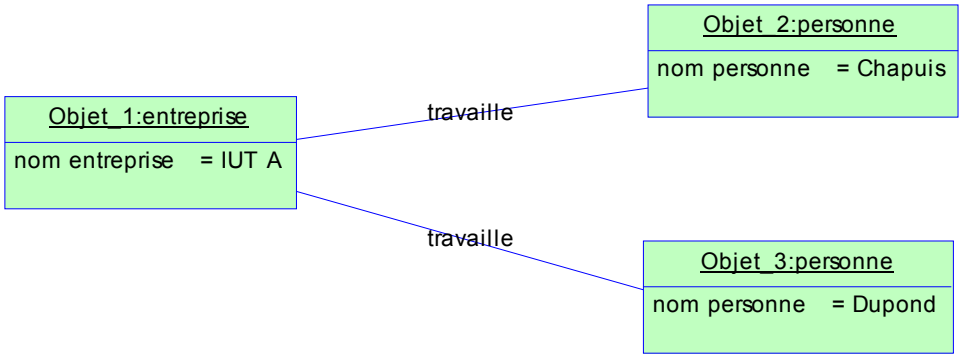
Une association peut être nommée. Le nom est une forme verbale, au milieu de la ligne qui symbolise l'association.

L'association est instanciable dans un diagramme d'objets ou de collaboration, sous forme de liens entre objets issus de classes associées

Notation :



Exemple d'instanciation

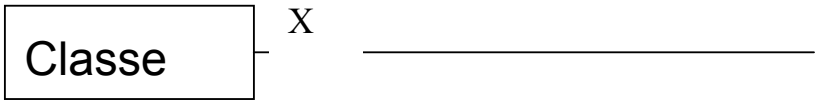


2.3.2 Composants et types d'association

Cardinalités ou Multiplicités

La multiplicité ou cardinalité est indiquée pour chaque extrémité ou rôle d'associations.

Elle indique pour une instance d'une classe, le nombre d'instances d'une autre classe qui peuvent lui être liées



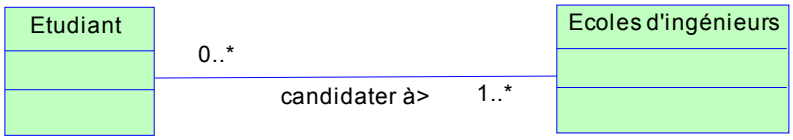
1 ou 1..1	exactement 1 (par défaut) exprime une contrainte de totalité
0..1	optionnel
n..m	de "n" à "m" (entiers naturels ou variables, m ≥ n)
n..*	de **n** à plusieurs (n, entier naturel ou variable)
0..*ou *	de 0 à plusieurs (équivalent **à x : *)**

Remarque :

Les cardinalités s'exprime dans un ordre inverse à celui du modèle Entité - association

Exemples:

1 - Un étudiant en IUT projette de poursuivre ses études



Une école d'ingénieur peut ne pas avoir de candidats, mais peut en avoir plusieurs. Un étudiant qui apparaît dans le SI a fait au moins une demande de poursuite d'études mais peut aussi en avoir fait plusieurs

2 - Gestion bancaire

–Un client est titulaire d'un ou plusieurs comptes; il peut effectuer des virements entre ses comptes,

– Un compte peut avoir jusqu'à 4 titulaires distincts,

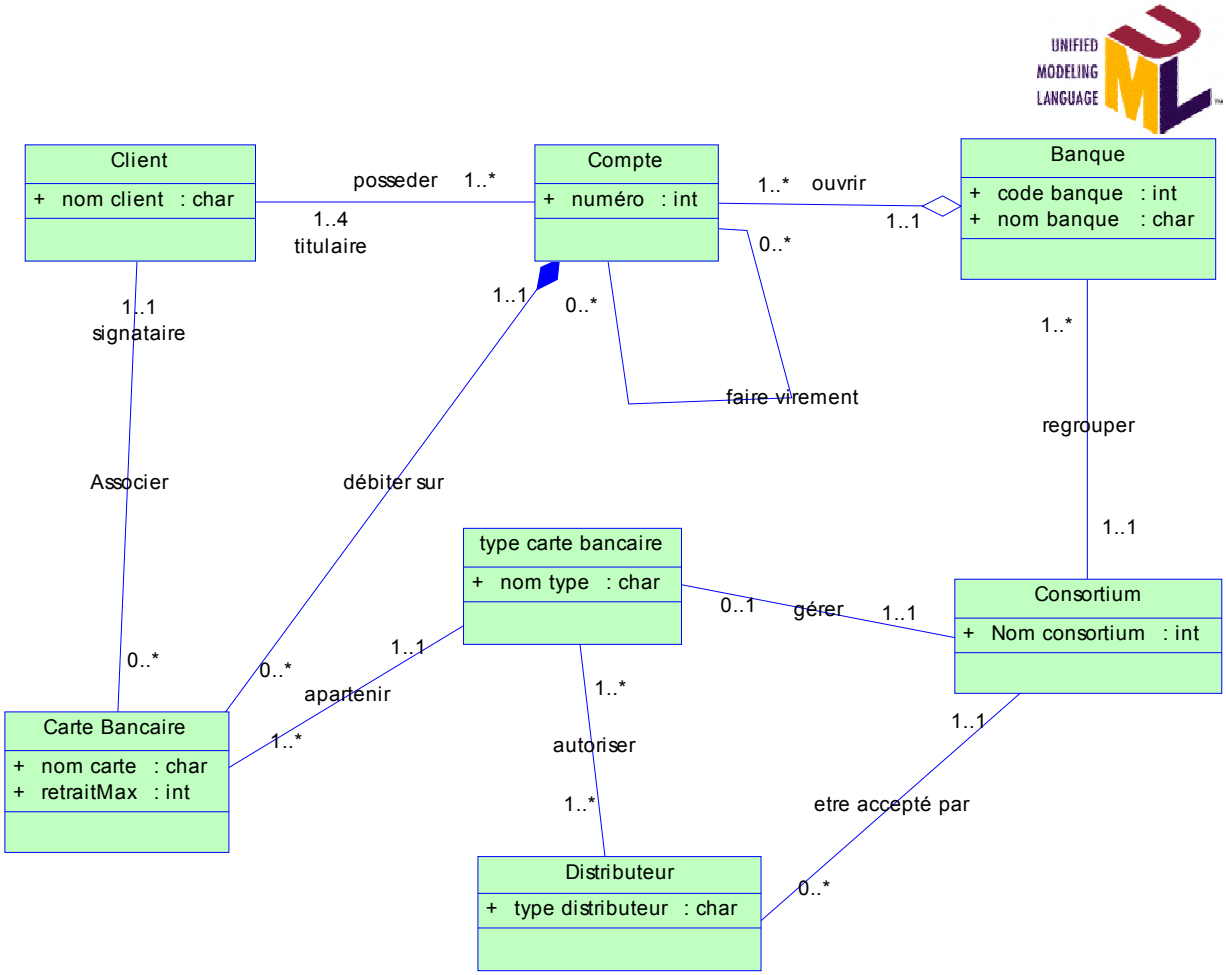
– Un compte peut avoir plusieurs CB associées,

– Un client signataire unique est associé à la carte bancaire délivrée sur un compte

– Certaines banques sont regroupées en consortium, certains de ces consortium gèrent un type de cartes bancaires. Chaque consortium possède ses propres distributeurs (GAB)

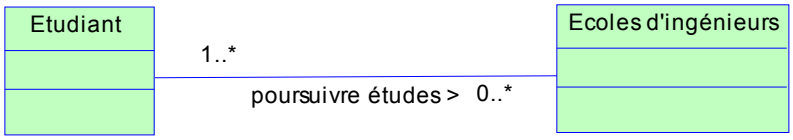
- Une carte bancaire appartient à un seul type. Un type de carte peut réunir plusieurs cartes (VISA normal, Visa Premier ,…)

- Un distributeur peut être utilisé par plusieurs types de carte. Un type de carte peut permettre de retirer de l'argent avec plusieurs distributeurs.



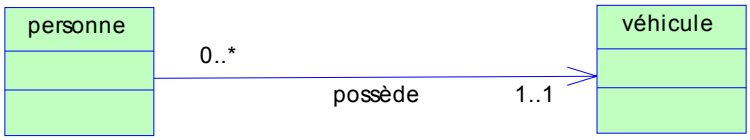
Association en forme verbale active :

Le sens de lecture principal du nom peut être précise par le symbole > dirigé vers la classe désignée par la forme verbale.



Association à navigabilité restreinte

Par défaut, une association binaire est navigable dans les deux sens. La réduction de la portée de l'association est souvent réalisée en phase d'implémentation, mais peut aussi être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre.



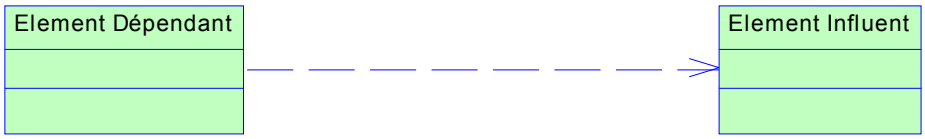
Navigation de personne vers véhicule uniquement.

Relation de dépendance :

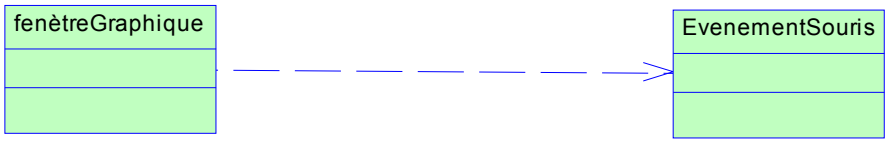
Une dépendance est une relation entre deux éléments de modélisation dans laquelle toute modification effectuée sur un élément de modélisation (*l'élément influent*) affecte l'autre élément (*élément dépendant*).

La relation de dépendance indique qu'un objet d'un diagramme utilise les services ou utilitaires d'un autre objet. On peut également définir des dépendances entre un package et un élément de modélisation.

Notation : flèche en pointillée



Exemple :



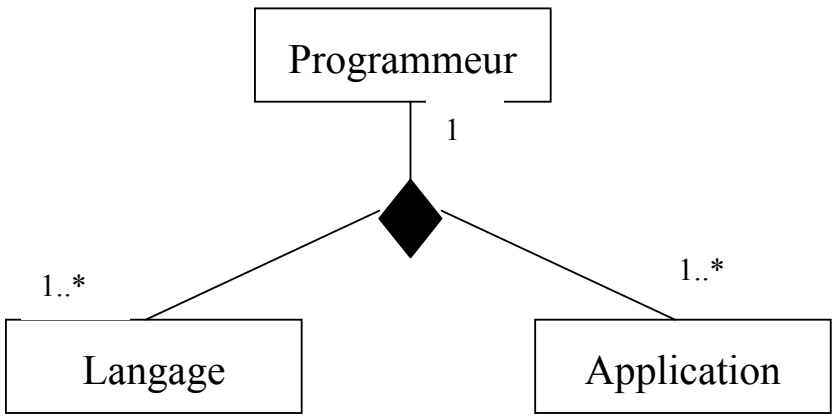
Association n-aire :

Il s'agit d'une association qui relie plus de deux classes. Non implanté sur PowerAMC

Note : de telles associations sont difficiles à déchiffrer et peuvent induire en erreur. Il vaut mieux limiter leur utilisation, en définissant de nouvelles catégories d'associations

Exemple :

Un programmeur utilise un langage donné pour un projet informatique donné

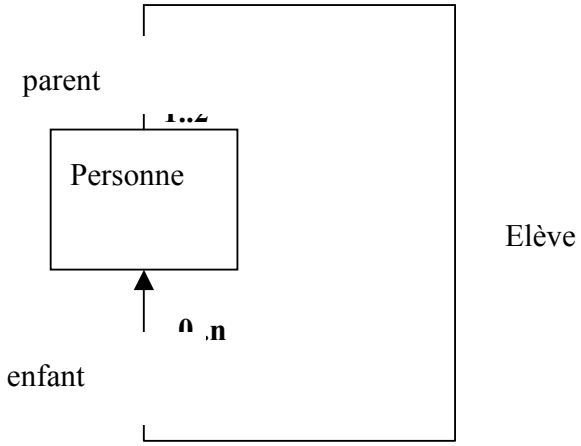


Nombre d'associations entre deux classes :

Il peut y avoir une, deux ou plusieurs associations de natures différentes entre deux classes.

Rôle

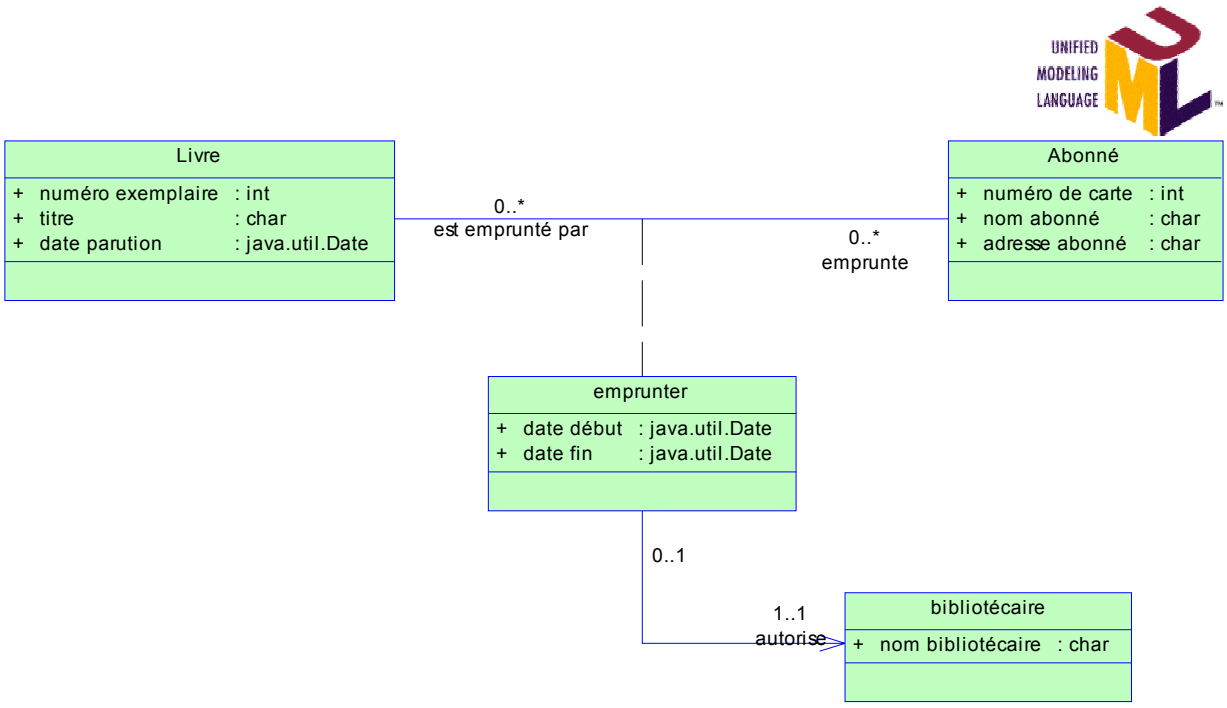
Spécifie la fonction d'une classe pour une association donnée (obligatoire pour les associations réflexives).



Classe d'association

Elle permet de décrire une association qui possède des attributs qui ne relèvent d'aucune classe en particulier mais bien du couple de classes.

Une classe d'association peut être en relation avec une autre classe.



2.4 Héritage

Les hiérarchies de classes permettent de gérer la complexité, en ordonnant les objets au sein d'arborescences de classes, d'abstraction croissante.

Une sous classe hérite des attributs, des méthodes et des associations liées à la super classe**.**

L'héritage peut être simple ou multiple.

Généralisation et spécialisation sont deux points de vue différents d'une démarche de conception d'héritage :

Spécialisation

Démarche descendante, Une classe est identifiée et de nouvelles classes sont créées pour spécifier les différences.

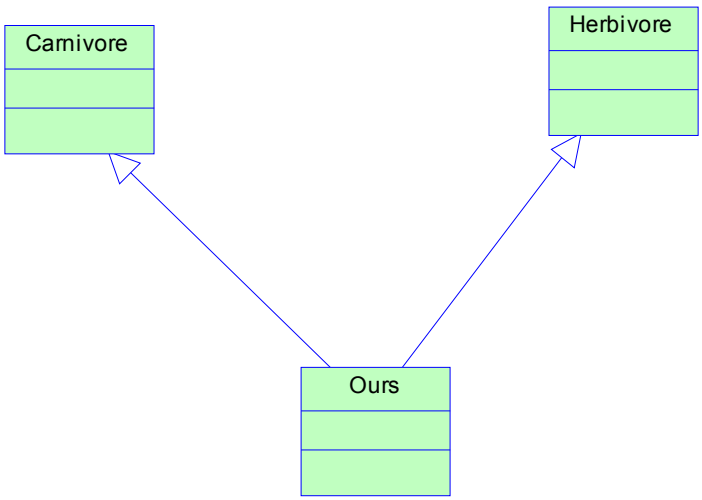
Consiste à étendre les propriétés d'une classe, sous forme de sous-classes, plus spécifiques.

Généralisation

Démarche ascendante, qui consiste à capturer les particularités communes d'un ensemble d'objets, issus de classes différentes.

Consiste à factoriser les propriétés d'un ensemble de classes, sous forme d'une super-classe.

La généralisation multiple existe dans PowerAMC : une sous classe peut avoir plusieurs classes génériques.



Classification

L'héritage (spécialisation et généralisation) permet la classification des objets.

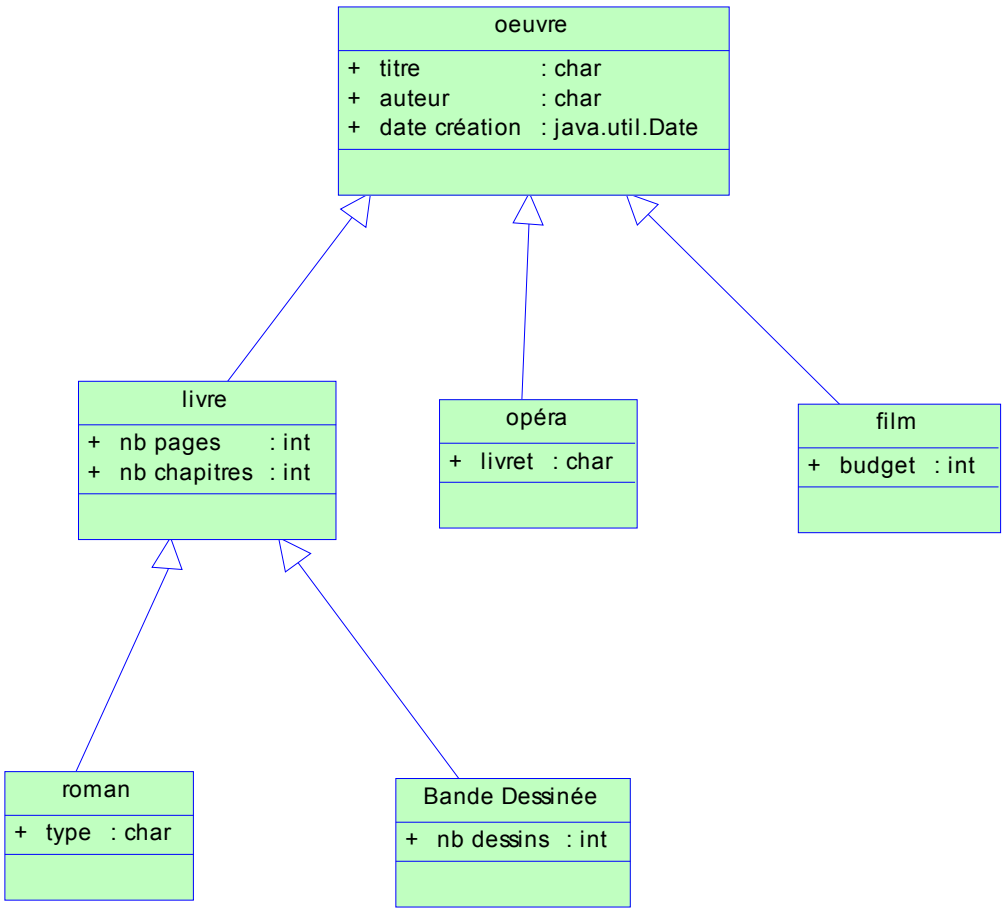
Une bonne classification est stable et extensible. Il ne faut pas classifiez les objets selon des critères instables (selon ce qui caractérise leur état) ou trop vagues (car cela génère trop de sous-classes).

Si Y hérite de X, cela signifie que "Y est une sorte de X" (analogies entre classification et théorie des ensembles).

L'héritage doit respecter le principe de substitution : En cas d'héritage, un objet d'une sous-classe peut, à tout moment, remplacer un objet de sa classe parente sans modifier le comportement du système.

Ne pas oublier que les critères de classification sont subjectifs.

Exemple



2.5 Agrégation

L'agrégation est une association non symétrique, qui exprime un couplage fort et une relation de subordination. Une des extrémité (le *composite ou conteneur ou agrégat*) joue un rôle prédominant par rapport à l'autre extrémité (l'*élément)*.

Une agrégation peut notamment (mais pas nécessairement) exprimer :

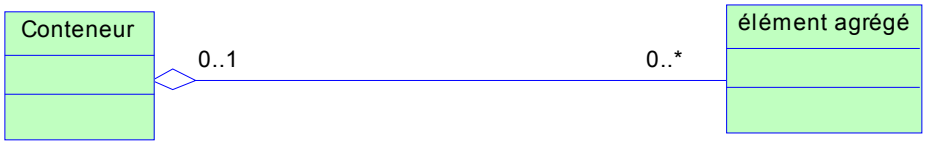
- qu'une classe (*élément)* fait partie d'une autre (*l'agrégat*),
- qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre classe,
- qu'une action sur une classe, entraîne une action sur une autre classe.

Le conteneur joue le rôle d'ensemble pour les instances de l'autre classe.

Une instance d'élément agrégé peut exister sans agrégat (et inversement) : les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.

Notation :

L'agrégation est représentée par un losange vide du coté du conteneur.

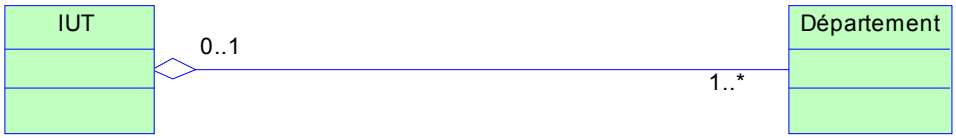


Remarque :

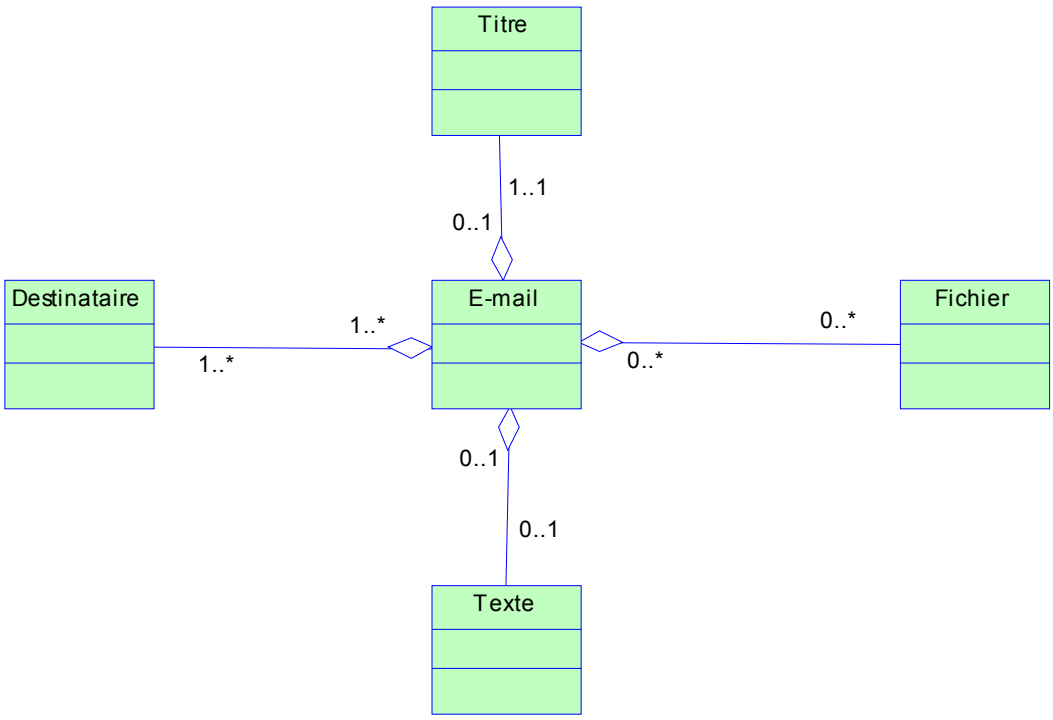
L'élément agrégé peut être associé à plusieurs conteneurs.

Exemples

1 – Un IUT est composé de départements, mais un département peut exister sans appartenir à un IUT



2 – Un mail contient un titre et peut contenir un ou plusieurs destinataires, et / ou un texte, et / ou un ou plusieurs fichiers attaché. Un même destinataire peut être associé à plusieurs e-mail. Un fichier pet être attaché à plusieurs e-mail.



2.6 composition

La composition est une agrégation forte.

Les cycles de vie des éléments et de l'agrégat (*composite*) sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.

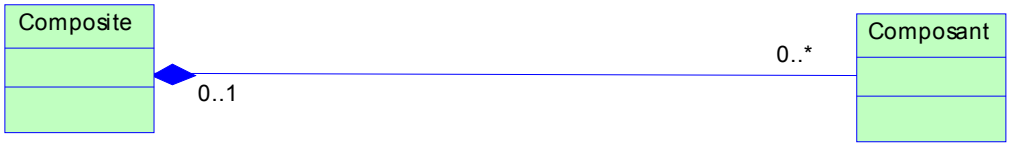
A un même moment, une instance d'élément ne peut être liée qu'à un seul agrégat.

Pour qu'il y ait composition il faut vérifier les critères suivants :

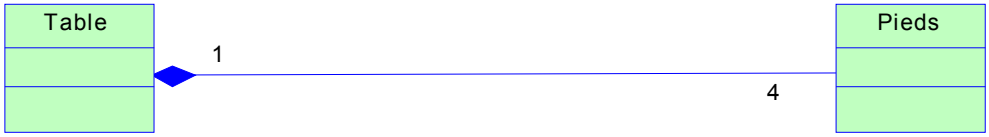
- La multiplicité ne doit pas être supérieure à 1 du coté du composite
- Le cycle de vie des élément dépend de celui du composite

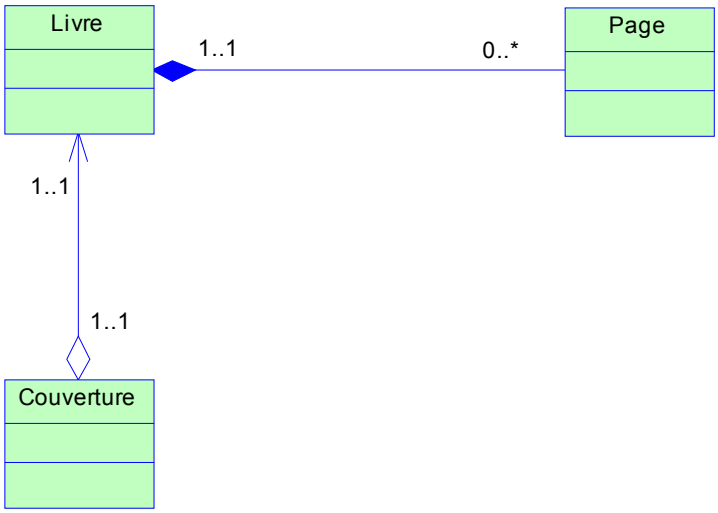
Notation :

La composition est représentée par un losange plein du coté du composite.



Exemples





Composition : Les pages sont physiquement contenues dans le livre,

Agrégation : Le livre peut bénéficier d'une couverture

2.7 Contraintes sur associations

Les contraintes sont des expressions qui précisent le rôle ou la portée d'un élément de modélisation (elles permettent d'étendre ou préciser sa sémantique). Une contrainte porte sur les instances de la classe, elle est placée sur un rôle.

Par exemple :

- {ordered} ou {ordonné}

  Indique que l'ordre des objets d'une classe doit être maintenu lors de l'ajout ou de la suppression d'objets.

  Rq : rien sur comment ça sera ordonné (choix conception)

- {frozen} ou {gelé}

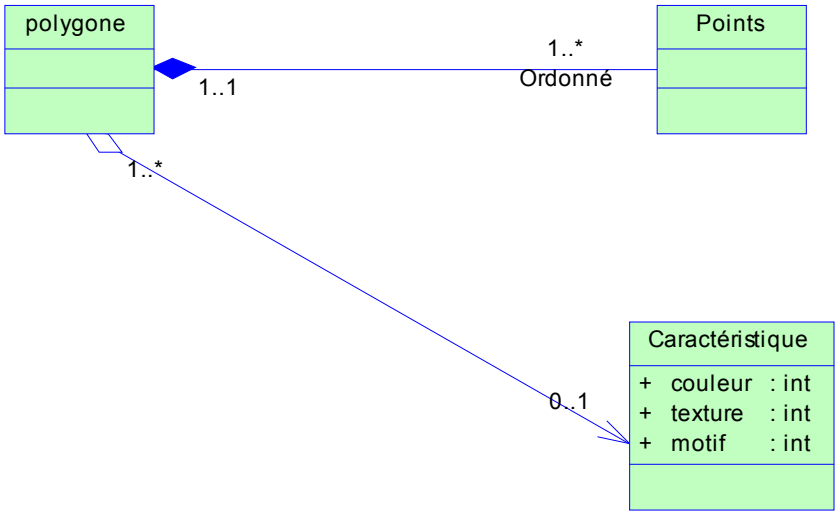
  –Un lien ne peut plus être modifié ni détruit après sa création

- {addOnly}

  –On ne peut qu'ajouter un objet, pas le détruire

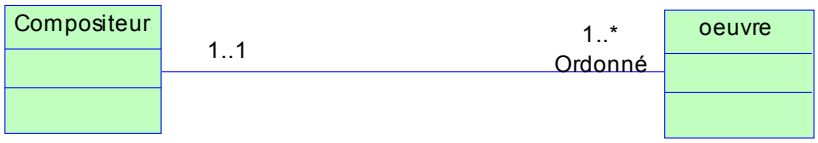
Exemples

1 - Un polygone contient au moins 3 points. C'est un agrégat de points ordonnés. Il peut être composé d'une caractéristique



2 - Relation entre œuvre et compositeur

On désire mettre en évidence l'ordre de la création des œuvre du compositeur.



Les contraintes peuvent s'exprimer soit en langage naturel, sous forme d'un texte encadré d'accolades, soit au moyen du langage OCL (Object Constraint Langage)

2.8 Encapsulation

Le principe général de l'encapsulation est de permettre l'accès aux données uniquement via les méthodes. Au niveau d'une implantation en bases de données relationnelles, une méthode sera une fonction, une procédure ou un déclencheur défini au niveau d'une table et agissant sur les données d'une ou plusieurs tables.

Les méthodes d'une classe représentent les *services* que fournit la classe en question aux autres classes. Une classe qui contient des méthodes est appelée *classe fournisseur*.

On peut distinguer trois types de classes :

- Classe décrite uniquement avec des attributs
- Classe décrites avec des attributs et des méthodes

- Classe décrite uniquement avec des méthodes

Positionnement des méthodes

Concernant le positionnement des méthodes, il n'y a pas de principe général d'établi.

Une méthode exprime un traitement que l'on peut réaliser sur ou avec un objet. Dans le cadre de l'analyse, il ne faut exprimer que les "opérations métiers", il est inutile de surcharger la description des classes par les méthodes de base comme les opérations constructeur / destructeur, accesseur, gestion des liens entre objets.

Remarque :

Un constructeur est un type d'opération particulier qui crée et initialise une instance de classe.

Un destructeur complète un constructeur en ce sens qu'il s'agit d'une opération qui détruit une instance de classe.

Un accesseur est une opération particulière qu'il est possible de créer pour un attribut de classe (renvoie de la valeur d'un attribut, chargement d'une valeur dans un attribut).

Syntaxe UML d'une Méthode

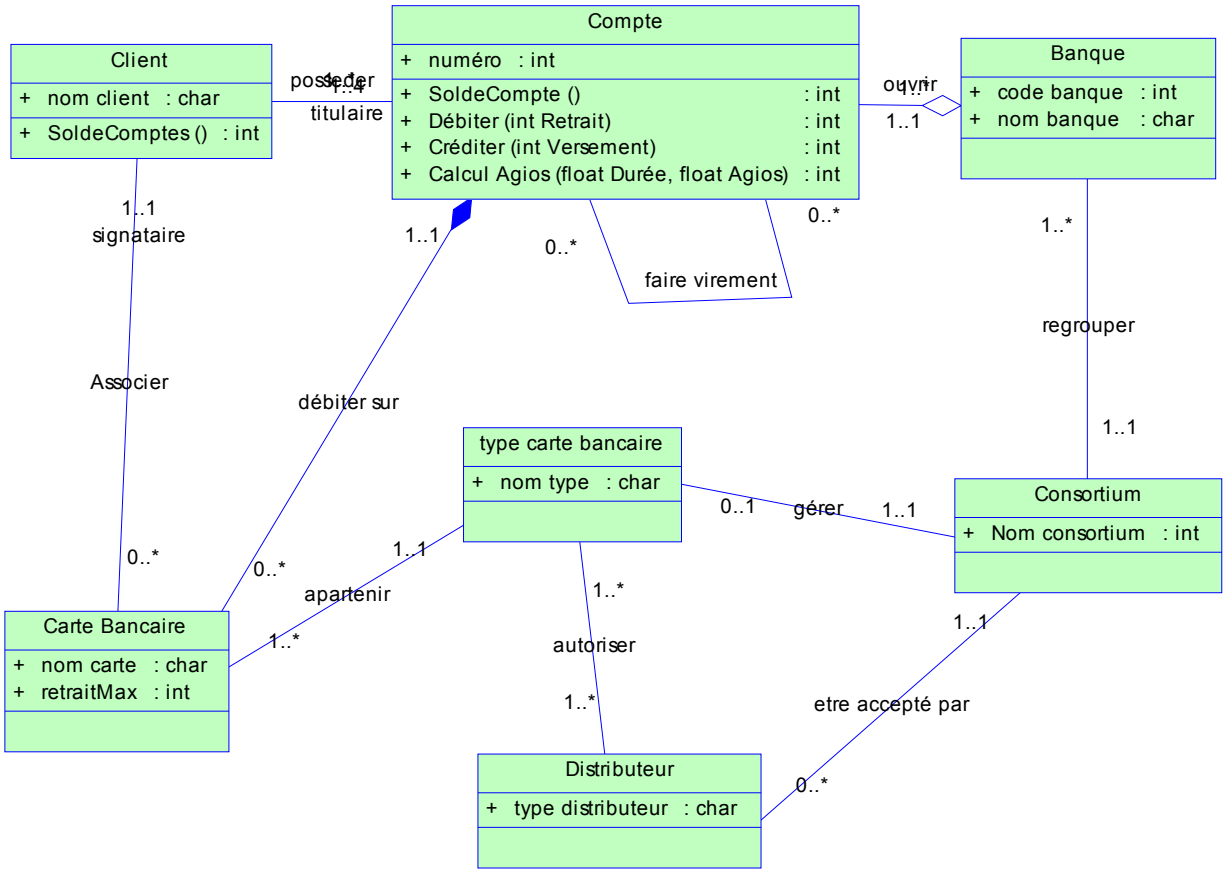
[visibilité] nom [(liste paramètres)] [:type_retour] [{propriété}]

La signature d'une méthode est constituée du nom de l'opération et éventuellement des paramètres d'appel.

UML définit trois niveaux de visibilité :

- Public (+) : rend la méthode visible de toutes les autres classes.
- Protégé (#) : rend la méthode visible uniquement aux sous classes de la classe.
- Privé (-) : rend la méthode visible à la classe seule.

Exemple


Client
+ nom client : char
+ SoldeComptes () : int
Compte
+ numéro : int
+ SoldeCompte () : int
+ Débiter (int Retrait) : int
+ Créditer (int Versement) : int
+ Calcul Agios (float Durée, float Agios) : int
Banque
+ code banque : int
+ nom banque : char
posséder
titulaire
1..4
1..1
ouvrir
1..1
1..1
signataire
1..1
0..*
faire virement
0..*
1..*
regrouper
Associer
débiter sur
type carte bancaire
+ nom type : char
0..1
gérer
1..1
Consortium
+ Nom consortium : int
1..1
0..*
0..*
1..1
apartenir
1..*
1..*
autoriser
1..*
Carte Bancaire
+ nom carte : char
+ retraitMax : int
1..1
etre accepté par
Distributeur
+ type distributeur : char
0..*